1. What Streamlit Is

Streamlit is a Python framework for building interactive web applications — mainly for data visualization, dashboards, and machine learning demos. You write Python code, and Streamlit automatically handles the UI, server, and rendering. No HTML, CSS, or JavaScript required.

Example use cases:

- Data dashboards and analytics tools
- Interactive machine learning model demos
- Exploratory data apps for internal teams

---

2. Installing Streamlit

```
pip install streamlit
```

To verify installation:

```
streamlit hello
```

This launches a sample app in your browser (usually at `http://localhost:8501`).

---

3. Your First App

Create a file named `app.py`:

```python
import streamlit as st

st.title("My First Streamlit App")
st.write("Hello, Streamlit!")
```

Run it:

```
streamlit run app.py
```

You'll see your app open automatically in your browser.

---

4. Core Building Blocks

Text and Display

```
st.title("Dashboard Title")
st.header("Section Header")
st.subheader("Subsection")
st.text("Plain text output")
st.markdown("**Markdown** works too")
st.code("print('Python code block')", language='python')
```

Data Display

```
import pandas as pd

data = pd.DataFrame({
    'x': [1, 2, 3],
    'y': [10, 20, 30]
})
st.dataframe(data)
st.table(data)
st.line_chart(data)
st.bar_chart(data)
```

---

5. Widgets (User Input)

Widgets let users interact with your app.

```
name = st.text_input("Enter your name")
age = st.slider("Select your age", 0, 100, 25)

if st.button("Submit"):
    st.write(f"Hello {name}, you are {age} years old!")
```

Other widgets include:

- `st.checkbox`
- `st.radio`
- `st.selectbox`
- `st.multiselect`
- `st.date_input`
- `st.file_uploader`

---

6. Control Flow and State

Streamlit reruns the entire script every time a widget changes. To keep state between runs, use `st.session_state`.

```
if 'count' not in st.session_state:
    st.session_state.count = 0

if st.button("Increment"):
    st.session_state.count += 1

st.write("Counter:", st.session_state.count)
```

---

7. Layouts and Organization

Columns

```
col1, col2 = st.columns(2)
col1.write("Left column")
col2.write("Right column")
```

Tabs

```
tab1, tab2 = st.tabs(["Data", "Chart"])
with tab1:
    st.write("Show data here")
with tab2:
    st.line_chart(data)
```

Sidebar

```
st.sidebar.title("Controls")
option = st.sidebar.selectbox("Choose an option", ["A", "B", "C"])
st.write("You chose:", option)
```

---

8. File Uploads and Downloads

```
uploaded_file = st.file_uploader("Upload a CSV", type=["csv"])
if uploaded_file is not None:
    df = pd.read_csv(uploaded_file)
    st.write(df)

st.download_button("Download CSV", df.to_csv().encode('utf-8'), "data.csv")
```

---

9. Caching for Performance

Use `@st.cache_data` to avoid recomputation of expensive functions:

```python
@st.cache_data
def load_data():
    return pd.read_csv("large_file.csv")

df = load_data()
```

There's also `@st.cache_resource` for caching models or other heavy objects.

---

10. Deployment

You can deploy a Streamlit app in several ways:

1. **Streamlit Community Cloud** (free): https://share.streamlit.io
2. **Custom hosting**: use `Docker`, `Heroku`, `AWS`, etc.
3. **Local servers** for internal dashboards.

Example (Dockerfile):

```dockerfile
FROM python:3.10
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
EXPOSE 8501
CMD ["streamlit", "run", "app.py", "--server.port=8501", "--
server.address=0.0.0.0"]
```

---

11. Best Practices

- Keep heavy computations outside the main script or cache them.
- Use `st.session_state` to manage interactivity cleanly.
- For complex apps, split into multiple pages (`pages/` directory).
- Avoid blocking operations (e.g., long model training) — consider async execution.
- Use environment variables for secrets and API keys.

---

12. Learning Resources

- Official Docs: https://docs.streamlit.io
- Gallery of apps: https://streamlit.io/gallery
- GitHub: https://github.com/streamlit/streamlit

End to End Sample Application

Simulating Chat GPT

This example will show how to:

1. Set up the Gemini API
2. Create a Streamlit interface with user input and chat history
3. Call the Gemini model and display responses

---

1. Install Dependencies

Run this in your terminal:

```
pip install streamlit google-generativeai python-dotenv
```

---

2. Get a Gemini API Key

Go to https://makersuite.google.com/app/apikey and create an API key. Store it safely, and don't hardcode it in your script.

Create a `.env` file in the same directory:

```
GEMINI_API_KEY=your_api_key_here
```

---

3. The Streamlit App (`app.py`)

Here's a complete working example:

```python
import streamlit as st
import google.generativeai as genai
import os
from dotenv import load_dotenv

# Load environment variables
load_dotenv()
genai.configure(api_key=os.getenv("GEMINI_API_KEY"))

# Initialize chat model
model = genai.GenerativeModel("gemini-1.5-flash")

# Streamlit app setup
st.set_page_config(page_title="Chat with Gemini", page_icon="💬")
st.title("💬 Chat with Gemini")
```

```
# Initialize chat history in session_state
if "messages" not in st.session_state:
    st.session_state.messages = []

# Display chat history
for msg in st.session_state.messages:
    role = "🧑‍💻 You" if msg["role"] == "user" else "🤖 Gemini"
    st.markdown(f"**{role}:** {msg['content']}")

# User input
prompt = st.chat_input("Type your message...")

if prompt:
    # Save user message
    st.session_state.messages.append({"role": "user", "content": prompt})
    st.markdown(f"**🧑‍💻 You:** {prompt}")

    # Send to Gemini
    with st.spinner("Gemini is thinking..."):
        chat = model.start_chat(history=st.session_state.messages)
        response = chat.send_message(prompt)
        reply = response.text

    # Save and display model response
    st.session_state.messages.append({"role": "model", "content": reply})
    st.markdown(f"**🤖 Gemini:** {reply}")
```

---

4. Run the App

From your terminal:

```
streamlit run app.py
```

This opens the app at http://localhost:8501.

You'll see:

- A title ("Chat with Gemini")
- A chat interface with history
- Input box for your message
- Streaming responses from Gemini

---

5. Optional Improvements

You can easily enhance it:

- **Add message bubbles** with Markdown styling:

```
st.chat_message("user").markdown(prompt)
st.chat_message("assistant").markdown(reply)
```

- **Add system prompt / personality**

```
system_prompt = "You are a helpful assistant specialized in Python and data
science."
model = genai.GenerativeModel("gemini-1.5-flash",
system_instruction=system_prompt)
```

- **Persist history** in a file (e.g., `pickle` or `json`) if you want sessions to survive reloads.
- **Upgrade model**:

```
model = genai.GenerativeModel("gemini-1.5-pro")
```

---

6. Example Folder Structure

```
project/
│
├── app.py
├── .env
├── requirements.txt
└── README.md
```

Example `requirements.txt`:

```
streamlit
google-generativeai
python-dotenv
```

---

7. How It Works (Step-by-Step)

1. User types a message.
2. Streamlit captures it and stores it in `st.session_state.messages`.
3. The app sends the user's prompt (plus chat history) to Gemini.
4. Gemini responds with text.
5. The message is displayed and stored for the next turn.
